AD-A068 390 UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES DEPT O--ETC F/G 12/2
EFFICIENT FRONTIERS FOR NONLINEAR MULTIDIMENSIONAL KNAPSACK PRO--ETC(U)
JAN 79 J S YORMARK N00014-75-C-0733
UNCLASSIFIED USC-WP-2 NL

END
DATE FILMED
6-79
DDC

LEVEL II

GRADUATE SCHOOL OF BUSINESS ADMINISTRATION

AND

SCHOOL OF BUSINESS

UNIVERSITY OF SOUTHERN CALIFORNIA

EFFICIENT FRONTIERS FOR NONLINEAR MULTIDIMENSIONAL KNAPSACK PROBLEMS

by

Jonathan S. Yormark
Department of Finance and Business Economics

Working Paper No. 2

March, 1979

This research was supported in part by the Office of Naval Research under Contract N00014-75-C-0733, Task NR042-323 Code 436. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ABSTRACT

An efficient recursive enumeration procedure is described for finding the complete family of efficient, or undominated, solutions for general separable multidimensional "knapsack" problems. A generalization of the Gilmore-Gomory procedure provides an alternative view of the Morin-Marsten method that admits an algorithm requiring significantly less computer storage and computational overhead. We illustrate the procedure with a highly nonlinear example.

1. Introduction

In this paper we develop an algorithm for generating the complete set of efficient, or undominated, solutions for the general separable nonlinear integer programming problem:

$$(P) \qquad \max \{r(x): g(x) \leq b, \ \underline{u} \leq x \leq \bar{u}, \ x \text{ integer}\},$$

where x, the vector of lower bounds $\underline{u}$, and the vector of upper bounds $\bar{u}$ are all n x 1, and b is m x 1. The return function $r(x) = \sum_{j \in J} r_j(x_j)$, $J = \{1, 2, \ldots, n\}$, and the resource consumption vector $g(x) = [g_1(x), g_2(x), \ldots, g_m(x)]'$, where each $g_i(x) = \sum_{j \in J} g_{ij}(x_j)$, $i \in I = \{1,2,\ldots,m\}$. We require only that the consumption functions satisfy the condition:

$$g_{ij}(x_j) \geq g_{ij}(\underline{u}_j) \text{ for all } i \text{ and } j. \tag{1.1}$$

No assumptions on the nature of the return functions $r_j(x_j)$ are required. We refer to condition (1.1) as the knapsack condition.

Any feasible solution x is said to be inefficient or dominated if there exists an alternative feasible solution $\tilde{x}$ such that:

$$r(\tilde{x}) \geq r(x) \tag{1.2}$$

$$\text{and} \qquad g(\tilde{x}) \leq g(x), \tag{1.3}$$

with at least one strict inequality holding in (1.2) and (1.3). Then $\tilde{x}$ is said to dominate x. Otherwise, $\tilde{x}$ and x are said to be mutually undominated. If (1.2) and (1.3) are satisfied with strict equality, then $\tilde{x}$ and x are said to be equivalent.

Alternative approaches for solving numerous variants and specialized sub-classes of (P) are widely represented in the literature. We use as a point of departure here, however, the excellent paper by Morin and Marsten [18]. The Morin-Marsten paper was the first to provide a unified framework demonstrating that the "curse of dimensionality" commonly attributed to dynamic programming methods can be successfully mitigated in certain broad

classes of multidimensional combinatorial optimization problems. Although more efficient methods for solving (P) exist [16], the procedure in [18] is the only parametric methodology available for generating the entire set of efficient solutions. An excellent review of the utility of the numerous special forms of (P) in a wide range of thoretical and applied problems in linear and integer programming can also be found in [18]; the reader is referred there for details. We do note, however, that some interesting non-separable variants of (P) have also appeared with particular applications in finance [8].

In §2 we describe a set of simplifying transformations which will reduce (P) to an equivalent problem in which the $g_{ij}(\cdot)$ and $r_j(\cdot)$ functions are everywhere nonnegative. This extends the Morin-Marsten procedure to a broader class of problems. In §3 we utilize a recursive enumeration approach to provide a significantly simpler derivation of the basic procedure of [18], and provide proofs of the fathom by infeasibility and fathom by dominance rules that do not appear in the original paper. In §4 we describe a computer implementation requiring significantly less storage, and the maintenance and manipulation of far less bookkeeping machinery, than that presented in [18]. This can be particularly important when m and n are of realistic size, since the efficient set is then normally quite large. An illustration of the implementation is presented in §5.

2. Simplifications

The algorithm we shall develop requires only condition (1.1). However, we shall find it useful to make certain transformations which do not in any way limit the generality of this procedure, but admit a significantly simpler derivation of the important results. Consider the standard technique for transforming lower bound restrictions into nonnegativity constraints. The usual transformation:

$$x_j' \equiv x_j - u_j \geq 0, \qquad j \in J \tag{2.1}$$

is used. Replace x_j by its equivalent $x_j' + u_j \geq 0$ wherever x_j appears. Note that x_j' may then be interpreted as the increment in excess of u_j associated with the value x_j.

If we re-write $g_{ij}(x_j)$ as $g_{ij}(x_j) = [g_{ij}(x_j) - g_{ij}(u_j)] + g_{ij}(u_j)$, the term in brackets may be viewed as the increment in resources consumed, in excess of $g_{ij}(u_j)$, due to an allocation x_j' in excess of u_j. Thus, we may denote this term as the incremental consumption funtion:

$$g_{ij}'(x_j') \equiv g_{ij}(x_j) - g_{ij}(u_j) \geq 0. \tag{2.2}$$

Similarly, under (2.1) we also obtain:

$$r_j(x_j) = r_j(x_j' + u_j) = [r_j(x_j) - r_j(u_j)] + r_j(u_j).$$

Denote the bracketed term as:

$$r_j'(x_j') \equiv r_j(x_j) - r_j(u_j), \qquad j \in J, \tag{2.3}$$

the incremental return over $r_j(u_j)$ for an incremental allocation x_j' over u_j.

Substituting (2.1) thru (2.3) into (P) yields the problem (P′):

$$\text{(P′)} \qquad \sum_{j \in J} r_j(u_j) + \max \sum_{j \in J} r_j'(x_j')$$

$$\text{s.t.)} \quad \sum_{j \in J} g_{ij}'(x_j') \leq b_i - \sum_{j \in J} g_{ij}(u_j), \qquad i \in I,$$

$$0 \leq x_j' \leq u_j'$$

$$x_j' \text{ integer}, \qquad j \in J,$$

where $u_j' = \bar{u}_j - u_j$.

Solving the equivalent problem (P′) solves (P), but now $g_{ij}'(x_j') \geq g_{ij}'(0) = 0$ for all i and j, and $r_j'(0) = 0$ for all j.

In addition, we may reduce the set of admissible $x_j \geq 0$ via the following three observations:

(i) If, for any $j \leq n$, there exists some $x_j = \ell > 0$ such that $r_j(\ell) \leq 0$, x_j need never equal ℓ in an undominated solution.

This follows from the observation that any solution containing $x_j = \ell > 0$ would always be dominated by, or equivalent to, the same solution with $x_j = \ell$ replaced by $x_j = 0$. Recognize that if we eliminate $x_j = \ell$ from consideration, the resulting reduced return function has $r_j(x_j) > 0$ for all $x_j > 0$.

(ii) If there exists any $j \leq n$ with x_j values ℓ and $\ell' \leq u_j$ such that:

$$r(\ell') \geq r(\ell)$$

and $$g_{ij}(\ell') \leq g_{ij}(\ell) \qquad \text{for all } i \in I,$$

then x_j need never equal ℓ in an undominated solution.

This follows from the fact that any solution containing $x_j = \ell$ is dominated by, or equivalent to, the same solution with $x_j = \ell$ replaced by $x_j = \ell'$. A reduction mechanism equivalent to (ii) is also provided in [18].

(iii) If, for any $j \leq n$, there exists some $x_j = \ell > 0$ such that $g_{ij}(\ell) > b_i$ for at least one $i \in I$, then no feasible solution containing $x_j = \ell$ exists.

In light of the foregoing, without loss of generality, we shall henceforth assume that:

$$r_j(x_j) \geq r_j(0) = 0 \text{ for } x_j = 0,1,\ldots,u_j, \tag{2.4}$$

and $$r_j(x_j) > r_j(0) \text{ whenever } x_j > 0,\ j \in J.$$

In addition,

$$g_{ij}(x_j) \geq g_{ij}(0) = 0 \text{ for } x_j = 0,1\ldots,u_j, \text{ and all } i \text{ and } j, \tag{2.5}$$

and there exists no x_j, $j \in J$, with a value $\ell \in \{1,2\ldots,u_j\}$ satisfying the conditions of (ii) or (iii) above.

Lastly, we observe that the value of $x_j = \ell$ may be viewed merely as an index denoting the appropriate resource vector

$$\underline{y}_j(\ell) = \underline{y}_j^{\ell} = [g_{1j}(\ell),\ g_{2j}(\ell),\ldots,\ g_{mj}(\ell)]' \tag{2.6}$$

consumed to receive a particular level of return $r_j(\ell)$ from variable j. Consequently, we may always redefine the associated index values for $x_j > 0$ so as to reorder the $\underline{y}_j(\ell)$ vectors, or the returns $r_j(\ell)$, into any convenient sequence. Recognize that (2.4) and (2.5) are unaffected. Thus, wherever convenient we may also assume, again without loss of generality, that:

$$0 = r_j(0) < r_j(1) \leq \cdots \leq r_j(u_j) \text{ for any } j = 1,2,\ldots,n. \tag{2.7}$$

The conditions of convenience in (2.4), (2.5), and (2.7) need not be explicit requirements in (P). We have shown that only the far weaker knapsack condition (1.1) is sufficient. Thus, the Morin-Marsten algorithm in [18], which assumes (2.4), (2.5), and (2.7) hold apriori, in fact solves a broader class of problems than originally specified.

3. Recursions

The derivation in this section is related to that utilized in [6], [11] and [17], and details an approach outlined in [16].

Define a set of n-vectors of the form $\underline{s}_k^{\ell} = (0,\ 0,\ldots,\ x_k,\ 0,\ldots,0)$ as:

$$\underline{S}_k \equiv \{\underline{s}_k^{\ell} : \ell=0,1,\ldots,\ u_k\} = \{x : x_k=\ell \in \{0,1,\ldots,u_k\},$$
$$\text{with } x_j=0,\ j\neq k,\ j\in J\}, \tag{3.1}$$
$$\text{for all } k = 1,2,\ldots,\ n.$$

Also, define a set of n-vectors of the form $\underline{x}_k^p = (x_1^p,\ x_2^p,\ldots,\ x_k^p,\ 0,\ldots,0)$ as:

$$\underline{X}_k \equiv \{\underline{x}_k^p : p = 0,1,\ldots,P_k\} = \{x : x_j \in \{0,1,\ldots,u_j\} \text{ for } j \leq k,$$
$$\text{with } x_j=0 \text{ for } j > k,\ j\in J\}, \tag{3.2}$$
$$\text{for all } k = 1,2,\ldots,n.$$

We let $\underline{X}_o \equiv \{\underline{x}_o^o\} = \{0\}$. The vector $\underline{x}_k^o = 0$ for all $k = 1,2,\ldots,n$. The value $P_k = [\prod_{j=1}^{k}(u_j + 1)] - 1$.

Given X_o, it is easy to see that

$$\begin{aligned} X_1 &= \{(0,0,\ldots,0),\ (1,0,\ldots,0),\ \cdots,\ (u_1,0,\ldots,0)\} \\ &= \{s_1^o + x_o^o,\ s_1^1 + x_o^o,\ldots,\ s_1^{u_1} + x_o^o\} \\ &= S_1 \oplus X_o, \end{aligned}$$

where the notation $\oplus$ denotes forming all possible sums of exactly one vector from S_1, and exactly one vector from X_o. If X_{k-1} is defined following (3.2), then

$$\begin{aligned} X_k &= S_k \oplus X_{k-1} \\ &= \{s_k^\ell + x_{k-1}^p \text{ for all } \ell = 0,1,\ldots,u_k \text{ and all } p = 0,1,\ldots,P_{k-1}\}. \end{aligned} \tag{3.3}$$

Note that $X_n = X \equiv \{x: 0 \le x \le u,\ x \text{ integer}\}$. Thus, recursive formula (3.3) provides a convenient method for generating all upper-bounded solutions $x \in X$. In addition, since $\{s_k^o\} \oplus X_{k-1} = X_{k-1}$, and $s_k^o \in S_k$ for all $k \le n$ so long as $u_k \ge 0$, we see that

$$X_k = S_k \oplus X_{k-1} = \{s_k^o\} \oplus X_{k-1} \cup S_k \oplus X_{k-1} = X_{k-1} \cup S_k \oplus X_{k-1}, \tag{3.4}$$

where $S_k = \{s_k^\ell: \ell = 1,2,\ldots,u_k\}$. This latter formula (3.4) is significantly more convenient for computer implementation. We need only enumerate the set of "augmentations" $S_k \oplus X_{k-1}$ added when generating X_k from X_{k-1}.

We may associate with each X_k the corresponding sets of resource consumption vectors B_k and returns R_k, derived from (3.3):

$$\begin{aligned} B_k &\equiv \{\beta_k^p: p=0,1,\ldots,P_k\} = \{g(x): x \in X_k\} = \{g(s_k^\ell): s_k^\ell \in S_k\} \oplus B_{k-1} \\ &= \{\gamma_k^\ell: \ell=0,1,\ldots,u_k\} \oplus B_{k-1} \\ &= B_{k-1} \cup \{\gamma_k^\ell: \ell=1,2,\ldots,u_k\} \oplus B_{k-1} \\ &= B_{k-1} \cup \Gamma_k \oplus B_{k-1}, \qquad \text{for all } k = 1,2,\ldots,n, \end{aligned} \tag{3.5}$$

where $\Gamma_k \equiv \{\gamma_k^\ell: \ell=1,\ldots,u_k\}$ and γ_k^ℓ is defined as in (2.6).
We define $B_o \equiv \{g(x): x \in X_o\} = \{g(x_o^o)\} = \{0\}$.

Similarly, $\underline{R}_k = \{r(x_k^p): p=0,1,\ldots,P_k\} = \{r(x): x \in \underline{X}_k\}$

$= \{r(s_k^\ell): s_k^\ell \in S_k\} \oplus R_{k-1}$

$= \underline{R}_{k-1} \cup \{r_k^\ell: \ell=1,\ldots,u_k\} \oplus R_{k-1}$,

for all $k = 1,2,\ldots,n$.

We define $\underline{R}_o = \{r(x_o^o)\} = \{0\}$. Also, $\underline{X}_k \subseteq \underline{X}$, $\underline{B}_k \subseteq \underline{B} = \{g(x): x \in \underline{X}\}$, and $\underline{R}_k \subseteq \underline{R} = \{r(x): x \in \underline{X}\}$ for all $k = 1,2,\ldots,n$.

Since we are interested only in solutions $x \in \underline{X}$ that are feasible for (P), we require a rule for eliminating infeasible items from each $\underline{X}_k$. As a result, we also reduce $\underline{B}_k$ and $\underline{R}_k$ accordingly. Define $X_k = \{x_k^p \in \underline{X}_k: g(x_k^p) \leq b, p=0,1,\ldots,P_k\}$, for $k=1,2\ldots,n$, and let B_k and R_k be the corresponding sets of consumption vectors and returns. Clearly $X_k \subseteq \underline{X}_k$, $B_k \subseteq \underline{B}_k$ and $R_k \subseteq \underline{R}_k$. And since $g(0) \leq b$, we have $X_o = \underline{X}_o = \{0\}$. Furthermore, $X_n = X = \{x: g(x) \leq b, o \leq x \leq u, x \text{ integer}\}$, with associated $B_n = B$ and $R_n = R$ defined accordingly. The following result provides the rule for determining which elements of $\underline{X}_k$ can be eliminated.

Proposition 1: (Feasibility test) Given $\beta_{k-1} \in B_{k-1}$ and $\gamma_k^\ell \in \Gamma_k$ for some $k=1,2,\ldots,n$, if $\beta = \gamma_k^\ell + \beta_{k-1} \not\leq b$, then $\beta \notin \underline{B}_k$. Furthermore, $\Gamma_{k+1} \oplus \{\beta\} \not\subseteq \underline{B}_{k+1}$, $\Gamma_{k+2} \oplus \Gamma_{k+1} \oplus \{\beta\} \not\subseteq \underline{B}_{k+2}$, ..., and $\Gamma_n \oplus \Gamma_{n-1} \oplus \cdots \oplus \{\beta\} \not\subseteq \underline{B}_n = \underline{B}$.

Proof: By assumption we have $\beta_{k-1} \in B_{k-1}$ and $\gamma_k^\ell \in \Gamma_k$. Suppose that $\beta = \gamma_k^\ell + \beta_{k-1} \not\leq b$ is generated. By definition $\beta \notin \underline{B}_k$. Now, consider $\Gamma_{k+1} \oplus \{\beta\}$, the set of all augmentations generated from β when generating B_{k+1}. Since $\gamma_{k+1}^\ell \geq 0$ for all $\ell > 0$, all such vectors $\beta + \gamma_{k+1}^\ell \not\leq b$ for every $\ell \in \{1,\ldots,u_{k+1}\}$. Thus, $\Gamma_{k+1} \oplus \{\beta\} \not\subseteq \underline{B}_{k+1}$. By a similar argument we see that $\Gamma_{k+2} \oplus \Gamma_{k+1} \oplus \{\beta\} \not\subseteq \underline{B}_{k+2}$, ..., and $\Gamma_n \oplus \Gamma_{n-1} \oplus \ldots \oplus \Gamma_{k+1} \oplus \{\beta\} \not\subseteq \underline{B}_n = \underline{B}$. ||

The proof of Proposition 1 also implies the following.

Corollary: If $\gamma_k^\ell + \beta_{k-1} \notin \underline{B}_k$, then the corresponding $\underline{x}_k = s_k^\ell + \underline{x}_{k-1} \notin \underline{X}_k$. Furthermore, $S_{k+1} \oplus \{\underline{x}_k\} \not\subseteq \underline{X}_{k+1}$, $S_{k+2} \oplus S_{k+1} \oplus \{\underline{x}_k\} \not\subseteq \underline{X}_{k+2}$, $\cdots$, and $S_n \oplus S_{n-1} \oplus \cdots \oplus S_{k+1} \oplus \{\underline{x}_k\} \not\subseteq \underline{X}_n = \underline{X}$.

Therefore,

$$X_k \subseteq S_k \oplus X_{k-1} \cup X_{k-1},$$

$$B_k \subseteq \Gamma_k \oplus B_{k-1} \cup B_{k-1}, \tag{3.6}$$

and $$R_k \subseteq \{r_k^{\ell}: \ell = 1,\ldots,u_k\} \oplus R_{k-1} \cup R_{k-1}.$$

The relationships (3.6) combined with Proposition 1 provide a mechanism for recursively generating all feasible solutions to (P), while also providing the associated set of returns and consumption vectors. The following theorem provides a recursive mechanism for identifying and eliminating vectors from X_k that are not efficient.

Let $X^* \subseteq X$ be the set of all undominated, or efficient, solutions for (P). Define $X_k^* = \{\underline{x}_k^p \in X_k \text{ that are undominated}\}$ at iteration $k = 1,2,\ldots,n$. Then $X_n^* = X^*$, and $X_o^* = X_o$; the zero-vector is always a member of the efficient set.

Proposition 2: (Elimination by Dominance)

Given $\underline{x}_k^p \in X_k$ and $\underline{x}_k^q \in X_k$ for some $k = 1,2,\ldots,n$, such that $r(\underline{x}_k^p) \geq r(\underline{x}_k^q)$ and $g(\underline{x}_k^p) \leq g(\underline{x}_k^q)$, then $\underline{x}_k^p$ dominates, or is equivalent to, $\underline{x}_k^q$. Consequently, $\underline{x}_k^q \notin X_k^*$, $S_{k+1} \oplus \{\underline{x}_k^q\} \subsetneq X_{k+1}^*$, ..., and $S_n \oplus S_{n-1} \oplus \cdots \oplus S_{k+1} \oplus \{\underline{x}_k^q\} \not\subseteq X_n^* = X^*$.

Proof: Let $\underline{x}_k^p$, $\underline{x}_k^q \in X_k$, and suppose that $r(\underline{x}_k^p) \geq r(\underline{x}_k^q)$ with $g(\underline{x}_k^p) \leq g(\underline{x}_k^q)$. By definition, $\underline{x}_k^p$ dominates, or is at worst equivalent to, $\underline{x}_k^q$ and therefore $\underline{x}_k^q \notin X_k^*$. Consider the sets $S_{k+1} \oplus \{\underline{x}_k^p\}$ and $S_{k+1} \oplus \{\underline{x}_k^q\}$, the augmentations to X_k generated from $\underline{x}_k^p$ and $\underline{x}_k^q$ at iteration k+1. Let $\underline{x}_{k+1}^{p'} = \underline{s}_{k+1}^{\ell'} + \underline{x}_k^p$, $\ell' \in \{1,\ldots,u_{k+1}\}$, be any particular candidate solution from $S_{k+1} \oplus \{\underline{x}_k^p\}$. Then there exists another solution $\underline{x}_{k+1}^{q'} = \underline{s}_{k+1}^{\ell'} + \underline{x}_k^q$. However, $r(\underline{s}_{k+1}^{\ell'}) = r_{k+1}^{\ell'} > 0$ and $\gamma_{k+1}(\underline{s}_{k+1}^{\ell'}) = \gamma_{k+1}^{\ell'} \geq 0$ for every possible value of ℓ'. Thus,

$$r(\underline{x}_{k+1}^{p'}) = r_{k+1}^{\ell'} + r(\underline{x}_k^p) \geq r_{k+1}^{\ell'} + r(\underline{x}_k^q) = r(\underline{x}_{k+1}^{q'})$$

and $$g(\underline{x}_{k+1}^{p'}) = \gamma_{k+1}^{\ell'} + g(\underline{x}_k^p) \leq \gamma_{k+1}^{\ell'} + g(\underline{x}_k^q) = g(\underline{x}_{k+1}^{q'}).$$

If $g(x^{p}_{k+1}) \not\leq b$, then $g(x^{q}_{k+1}) \not\leq b$. If x^{q}_{k+1} is infeasible, it is eliminated from X^{*}_{k+1} by definition. If $g(x^{q}_{k+1}) \leq b$, then $g(x^{p}_{k+1}) \leq b$. Thus, the entire set $S_{k+1} \oplus \{x^{q}_{k}\}$ is dominated by, or equivalent to, the set of solutions $S_{k+1} \oplus \{x^{p}_{k}\}$. Therefore, $S_{k+1} \oplus \{x^{q}_{k}\} \mathcal{I} X^{*}_{k+1}$. By a similar argument, $S_{k+2} \oplus S_{k+1} \oplus \{x^{q}_{k}\} \mathcal{I} X^{*}_{k+2}, \ldots$, and $S_{n} \oplus S_{n-1} \oplus \cdots \oplus S_{k+1} \oplus \{x^{q}_{k}\} \mathcal{I} X^{*}_{n} = X^{*}$. ||

In light of Proposition 2, x^{q}_{k} may be dropped from consideration at iterations k, k+1,...,n.

We may combine the above fathoming criteria into the algorithm outlined as follows:

0. Set k=0, and set $X_{o} = B_{o} = R_{o} = \{0\}$.

1. Set k=k+1. If k > n, stop.

2. Set $P = |B_{k-1}| - 1$, and label the elements of B_{k-1} as $\{\beta^{o}, \beta^{1}, \ldots, \beta^{P}\}$, with associated solutions $X_{k-1} = \{x^{o}, x^{1}, \ldots, x^{P}\}$ and returns $R_{k-1} = \{r(x^{o}), r(x^{1}), \ldots, r(x^{P})\}$. Set $B_{k} = B_{k-1}$, $X_{k} = X_{k-1}$, and $R_{k} = R_{k-1}$.

3. Let p = 0;

4. Let $\ell = 1$;

5. Let $x = s^{\ell}_{k} + x^{p}$, with corresponding payoff $r = r^{\ell}_{k} + r(x^{p})$, and consumptions $\beta = y^{\ell}_{k} + \beta^{p}$.

6. (Feasibility test) If $y^{\ell}_{k} + \beta^{p} \not\leq b$, go to step 9.

7. (Dominance tests)

   (i) If x is dominated by a solution currently in X_{k}, go to step 9.

   (ii) If x dominates any solution currently in X_{k}, this dominated solution may be fathomed.

8. Set $B_{k} = B_{k} \cup \{y^{\ell}_{k} + \beta^{p}\}$, $X_{k} = X_{k} \cup \{x\}$, and $R_{k} = R_{k} \cup \{r\}$.

9. Set $\ell = \ell + 1$. If $\ell \leq u_{k}$, go to step 5. Otherwise,

10. Set p = p + 1. If $p \leq P$, go to step 4. Otherwise, go to step 1.

At termination, $X_{n} = X^{*}$, the set of all undominated solutions for (P).

The outline provided above is virtually identical to the procedure outlined in §2 of [18], although the algorithm here has added flexibility in that the loops on p and ℓ may be interchanged if desired. However, our development is based on only the most fundamental concepts of recursive enumeration laid down by Bellman [1,2,3,4], Dantzig [5], Karush [12], Gilmore and Gomory [9], Kettelle [13] and Loane [15], utilized in the later work of Dreyfus and Prather [7], Greenberg [10], Lawler and Bell [14], Nemhauser and Ullman [19], Weingartner [20-App. A], Weingartner and Ness [21], and Yormark [22], and subsequently formalized and extended in [11], [16], [17], [18], and [23]. When m = 1 we have essentially the Gilmore-Gomory procedure of [9]. We call this type of derivation an imbedded solution space approach. It again makes clear that the dimensionality of the state space need not influence the algorithmic development. Moreover, this approach allows for simple proofs of Propositions 1 and 2, and also admits very simple proofs of the results required for the implementation strategy described in §4.

4. Implementation

Although the algorithm of §3 is of the same form as that in [18], our approach to identifying solutions that dominate the candidate is significantly different. The search in step 7 in our implementation is based on the following concept. Let $\underline{x}$ be any candidate solution derived from progenitor $\beta^p \in B_{k-1}$ at step 5 of the algorithm, with $\underline{\beta} = \gamma_k^{\ell} + \beta^p$. Define

$$\underline{h} \equiv \max_i \{\underline{\beta}_i\}. \tag{4.1}$$

Let β' be the resources associated with some alternative feasible solution $\underline{x}' \neq \underline{x}$, with associated maximum element h'.

Proposition 3: If $\beta' \leq \underline{\beta}$, then $h' \leq \underline{h}$.

Proof: By hypothesis, $\beta_i' \leq \underline{\beta}_i$ for all i=1,2,...,m. Then $\underline{h} \geq \underline{\beta}_i \geq \beta_i'$ for each i=1,2,...,m, and therefore, in particular, for that (not necessarily unique) $i = i^*$ for which $\beta_{i^*}' = h'$ obtains.||

Thus, any solution $\underline{x}'$ which can dominate the candidate $\underline{x}$ must have $h' \leq h$. And if $\underline{x}$ dominates some solution $\underline{x}' \neq \underline{x}$, then $h \leq h'$.

We exploit Proposition 3 by envisioning the elements of B_k (with their corresponding returns) to be in nondecreasing order of the associated h. The labeling in step 2 thus implies $h^p \leq h^{p+1}$, $p=0,1,\ldots,P$. We check for dominance at step 7(i) using the following strategy. Any solution $\underline{x}^q$ dominating $\underline{x}$ has $h^q \leq h$. Since $\gamma_k^\ell > 0$, we know that $h > h^p$, but $h^q \leq h^p < h$ for all $q < p$ also. We avoid examining $\underline{x}^q$ for all $q < p$, and maintain a tight bound on those $\underline{x}^q$ that might dominate the candidate, by keeping track of that index q for which

$$h^q = \min_{q>0} \{h^q : r(\underline{x}^q) = r^q > r^p = r(\underline{x}^p)\}$$

obtains. Then $r^q \leq r^p < r$ for all $q < q$, and any $\underline{x}^q$ with $q < q$ cannot dominate $\underline{x}$. Suppose $h^s \leq h < h^{s+1}$. By Proposition 3 only elements q, $q \leq q \leq s$, correspond to solutions which may dominate the candidate. The test of step 7(i) is applied only to such q; if $r^q \geq r$, we check to see if $\beta_i^q \leq \beta_i$ for all $i=1,2,\ldots,m$. This row-wise search is preempted as soon as some $\beta_i^q > \beta_i$. If $\underline{x}$ is undominated by all $q \leq q \leq s$, then $\underline{\beta}$ is "inserted" into B_k.

Once $\underline{\beta}$ is inserted (and $\underline{x}$ saved), we must execute step 7(ii) and examine those solutions possibly dominated by the newly-added $\underline{x}$. We avoid searching all the elements of B_k having $h^q \geq h$ due to the following result. (A similar observation appears in [18]).

Proposition 4: Any solution $\underline{x}^q$ which $\underline{x}$ dominates has an associated payoff r^q such that $r^p < r^q < r$.

Proof: Suppose $\underline{x}$ dominates $\underline{x}^q$ with corresponding $r^q \leq r^p$. Since β^q is in B_k, $\beta^p \not\geq \beta^q$. Then, although $r = r^p + r_k^\ell > r^q$, we still have $\beta = \beta^p + \gamma_k^\ell \not\geq \beta^q$. ||

We store the data necessary to execute the algorithm in a two-dimensional array B(I,J). Within column J, at completion of iteration k=1,2,...,n:

B(0,J) contains the return r^p associated with some solution $\underline{x}_k^p \in X_k$,

B(I,J), for I=1,2,...,m, contains the vector β^p associated with $\underline{x}_k^p$,

B(m+1,J) contains the h^p associated with β^p,

and B(m+2,J) contains the column number corresponding to h^{p+1}.

In addition,

B(m+3,J) contains the column number of that column in B(I,J) with the next largest return.

The link-list B(m+2,J) maintains the columns of the B-array in nondecreasing order of h without requiring them to actually be in any particular physical order in the computer memory, thereby circumventing the data-handling problems of inserting new vectors into B. Whenever $h^p = h^{p+1}$, we order such that $r^p \leq r^{p+1}$. That is,

$$B[m+1,p] = B[m+1,B(m+2,p)] \tag{4.2}$$

$$\text{implies} \quad B[0,p] \leq B[0,B(m+2,p)].$$

The link-list B(m+3,J) threads the columns of the B-array in nondecreasing order of r. If two undominated solutions have equal return,

$$B[0,p] = B[0,B(m+3,p)]$$

$$\text{implies} \quad B[m+1,p] \leq B[m+1,B(m+3,p)].$$

Setting B(m+2,J) or B(m+3,J) to 1 denotes that there are no successors in the associated sequence.

Storing X_k requires a linked-list version of the standard index-list common to dynamic programming implementations (c.f. [7], [9], [10], or [22]). We use the scheme described in detail in [18], and not repeated here.

We need not choose successive progenitors in the sequence implied by step 10 of §3. Having examined all descendants of some progenitor $\underline{x}_{k-1}^p \in X_{k-1}$, it is more efficient to use the link-list B(m+3,J) to find that $\underline{x}_{k-1} \in X_{k-1}$ with the next largest payoff. Then h^q becomes monotone nondecreasing over the

successive progenitors and updating q is trivial. In addition, given any particular value of $\ell > 0$ we may determine a still tighter bound for the dominance scan in step 7(i) by finding the index q^* such that

$$h^{q^*} = \min_{q \geq q} \{h^q : r^q \geq r^p + r_k^\ell\}.$$

If $h^{q^*} > \underline{h}$, then $\underline{x}$ is undominated and we need only update the link-list B(m+2,J). In general $h^{q^*} \leq \underline{h}$ and the data needed to update B(m+2,J) is obtained during the scan in step 7(i). Note that if the returns satisfy condition (2.7), h^{q^*} is monotone nondecreasing in ℓ and updating is still more efficient.

Following Proposition 4, in step 7(ii) we examine only solutions with payoffs between r^p and $\underline{r}$ by following the link-list B(m+3,J). Starting from the progenitor, we scan the elements of the B-arry in nondecreasing order of payoff, examining for dominance those with $h^q \geq \underline{h}$ and return $r^q \leq \underline{r}$. We stop immediately upon encountering an element q' whose return $r^{q'} > \underline{r}$. We then update B(m+3,J) without additional search.

This last procedure can be improved by recognizing that elements $q \leq s$ with $h^q < \underline{h}$, examined previously in step 7(i), need never be re-examined in step 7(ii). During the scan of step 7(i), we can easily keep track of that column $J = j^*$ corresponding to $\max \{r^q \leq \underline{r} : h^q < \underline{h} ; q \leq q \leq s\}$. We then initialize the scan in step 7(ii) at column B(m+3, j^*).

When $u_j = 1$ for all $j \leq n$, the entire algorithm is often more efficient if the variables are renumbered such that:

$$r_j(1)/\underline{h}^j \geq r_{j+1}(1)/\underline{h}^{j+1}, \quad j=1,2,\ldots,n-1, \tag{4.3}$$

where $\underline{h}^j \equiv \max_i \{g_{ij}(1)\}$. This is exactly analogous to the "bang-for-the-buck" ordering common to knapsack algorithms. When $m = 1$, of course, both orderings must be identical.

When $u_j > 1$, a natural extension of this idea is to re-order the ℓ-values, within a specified $j \leq n$, such that:

$$r_j(\ell)/h^j(\ell) \geq r_j(\ell+1)/h^j(\ell+1), \tag{4.4}$$

$$\ell=1,2,\ldots,u_j-1,$$

where $h^j(\ell) \equiv \max_i \{g_{ij}(\ell)\}$. Then, renumber the variables according to (4.3).

The use of $\underline{h}$ as an ordering criterion is an attempt to find a suitable surrogate for the vectors β that is both effective in identifying potentially dominating columns, but avoids examining individual components wherever possible. The ordering scheme embodied in Proposition 3, however, is surprisingly robust in the sense that there are alternative surrogates for β (see [23] for example) which will also satisfy the proposition. While (4.1) is always valid, an alternative is to define

$$\underline{h} \equiv \min_i \{\beta_i\}. \tag{4.5}$$

It is easy to see that the relationship in Proposition 3 is maintained under (4.5). Moreover, the entire implementation, and the ordering of variables via (4.3) and (4.4), remains exactly as described above, except that $\underline{h}$ is now computed using (4.5).

The rule (4.1) looks to insert the new candidate "far" from its progenitor p because the ordering is on the largest element in β. Thus, the scan in step 7(i) may become unnecessarily long. By ordering on the smallest element in $\underline{\beta}$ we keep these scans "short", since relatively fewer potentially dominating columns qualify under Proposition 3. Unfortunately, (4.5) need not be a uniformly better choice. In problems with a low density of nonzero entries in the consumption function tables, the $\underline{h}$ values under (4.5) are frequently zero. This can result in extended sequences of columns with equal $\underline{h}$ values of zero. Then, the dominance scans in step 7(i) can have more frequent, expensive row-wise comparisons. Rule (4.1) avoids this problem and is preferred in such cases. For problems with a low density of nonzero elements in the consumption function tables, however, (4.5) is usually supe-

rior to (4.1). Capital budgeting problems are a prime example of such a class. Many of the test problems listed in [18] are capital budgeting models.

The reader familiar with the Morin-Marsten procedure will recognize that the implementation here can be expected to be more efficient than that in [18] when m and n are of realistic size. The Morin-Marsten procedure maintains an array of m link-lists (called PERMUTE; see [18 - p.1152]), one for each constraint. These thread the corresponding β_i^p, p=0,1,...,P, in nondecreasing order. All m link-lists must be scanned over all β_i^q, $\beta_i^p < \beta_i^q \leq \beta_i$, every time a candidate is added to the B-list in order to update each row of PERMUTE for the next scan. Moreover, as m increases, this computational burden increases at least linearly. Our implementation maintains but a single link-list for the equivalent scan, regardless of the value of m. And updating of this list is implicit to the algorithm.

Finally, we note that each column of the B-array requires m+4 words of central memory, excluding additional locations needed to store the associated solutions. As dominated columns are dropped, all these locations can be re-used. Such "empty" columns are recycled in a FIFO manner, before the active length of B is extended, as in the implementation of [18]. However, the algorithm in [18] requires 2m+2 words of memory per column excluding locations needed only to save the undominated solutions. Consequently, (m-2/m+4)=75% more storage is required than with our approach when m is as small as 20. This can be important when the number of constraints, or the number of undominated strategies, is large.

5. An Example

To illustrate the implementation strategy of §4, we have developed a small, but highly nonlinear sample problem in n=3 variables with m=2 constraints. The raw data for the return and consumption functions are given

in Figure 1(a). Note that $\underline{u}=(1,2,0)$ and $\bar{u}=(5,5,3)$, with $b=(2,5\frac{1}{2})'$. After applying (2.1), we obtain the revised functions (2.2) and (2.3) listed in Figure 1(b). Note that now $\underline{u}' = (0,0,0)$, $\bar{u}' = (4,3,3)$, and $b' = (5, 5\frac{1}{2})'$. By (i) of §2, we may ignore $x_1' = 2$, and via (ii) of § 2 we may ignore $x_2' = 2$. Thus, in Figure 1(c) we have listed the equivalent functions with these columns deleted and the admissible x_j adjusted accordingly. In Figure 1(d) the data have been re-ordered according to (4.3) and (4.4).

At the end of iteration k=1, the B-array appears in main memory exactly as shown below. Column numbers and labels are for convenience only.

Row	1	2	3	4
0	0	4	2	2.5
1	0	3	2	1
2	0	2	2	3
3	0	3	2	3
4	3	1	4	2
5	3	1	4	2
	β_1^0	β_1^3	β_1^1	β_1^2

The array below is the B-array above listed according to the link list B(m+2,J) = B(4,J). Note the h values in B(m+1,J) are now in nondecreasing order.

Row	1	3	4	2
0	0	2	2.5	4
1	0	2	1	3
2	0	2	3	2
3	0	2	3	3
4	3	4	2	1
5	3	4	2	1
	β_1^0	β_1^1	β_1^2	β_1^3

	x_1					x_2				x_3				
	1	2	3	4	5	2	3	4	5	0	1	2	3	b
$r_j(x_j)$	1	5	-1	3½	3	1	4	2	3½	-1	2½	2	1½	
$g_{1j}(x_j)$	-1	2	3	0	1	0	2	2½	3	-2	-1	0	½	2
$g_{2j}(x_j)$	2	4	4	5	4	-1	2	3	1	-1	2½	1½	½	5½

(a) Original Data

	x_1'					x_2'				x_3'				
	0	1	2	3	4	0	1	2	3	0	1	2	3	b′
$r_j(x_j)$	0	4	-2	2½	2	0	3	1	2½	0	3½	3	2½	
$g_{1j}(x_j)$	0	3	4	1	2	0	2	2½	3	0	1	2	2½	5
$g_{2j}(x_j)$	0	2	2	3	2	0	3	4	2	0	3½	2½	1½	5½

(b) Data Under (2.1)

	x_1'				x_2'			x_3'				
	0	1	2	3	0	1	2	0	1	2	3	b′
$r_j(x_j)$	0	4	2½	2	0	2½	3	0	3½	2½	3	
$g_{1j}(x_j)$	0	3	1	2	0	3	2	0	1	2½	2	5
$g_{2j}(x_j)$	0	2	3	2	0	2	3	0	3½	1½	2½	5½
ratio (4.4)		1.3	.83	1.0		.83	1.0		1.0	1.0	1.2	

(c) Data Reduced Via Dominance

	x_1'				x_2'				x_3'			
	0	1	2	3	0	1	2	3	0	1	2	b′
$r_j(x_j)$	0	4	2	2½	0	3	2½	3½	0	3	2½	
$g_{1j}(x_j)$	0	3	2	1	0	2	2½	1	0	2	3	5
$g_{2j}(x_j)$	0	2	2	3	0	2½	1½	3½	0	3	2	5½
ratio (4.4)		1.3	1.0	.83		1.2	1.0	1.0		1.0	.83	

(d) Data Ordered By (4.3) and (4.4)

Figure 1. Data for the Example Problem

To illustrate a major cycle, consider interation k=2 at the point where we are about to enter step 6 of the algorithm with the progenitor being β_1^2 and $\ell = 2$. The progenitor β_1^2 is stored in column 4. The corresponding B-array is listed below in h order, along with the candidate $\beta = \gamma_2^2 + \beta_1^2$ with payoff r = 5.

Row	1	3	6	5	4	2	7	9	8	10			
0	0	2	2.5	3	2.5	4	3.5	4.5	5	5.5			
1	0	2	2.5	2	1	3	1	4.5	4	3			
2	0	2	1.5	2.5	3	2	3.5	3.5	4.5	5.5			
3	0	2	2.5	2.5	3	3	3.5	4.5	4.5	5.5			
4	3	6	5	4	2	7	9	8	10	1			
5	3	6	4	7	5	9	2	8	10	1			
	β_1^0	β_1^1			β_1^2	β_1^3							
	β_2^0	β_2^1	β_2^2	β_2^3	β_2^4	β_2^5							

5	= r
3.5 4.5	= β
4.5	= h

We refer to this array in the following discussion.

From the prior cycle with β_1^1, the value $h^q = 2.5$ corresponded to column 6. Since the payoff in column 6 is now not greater than the return found in the β_1^2 column, we search "to the right" from column 6, finding that the updated h^q is in column 5. We also note that the candidate is feasible. We thus search "to the right" from column 5 stopping at column 8, the column corresponding to $h^{q^*} = 4.5$; column 8 contains the "leftmost" potentially dominating solution. All other columns "between" 5 and 8, with h values no larger than $\underline{h} = 4.5$, are eliminated by value. Examining column 8 more closely, we test the elements row-wise against $\underline{\beta}$. Since $B(1,8) = 4 > \underline{\beta}_1 = 3\frac{1}{2}$,

we may immediately conclude that column 8 cannot dominate the candidate, and preempt the check of row 2. Furthermore, since $B(m+1,10) = 5\frac{1}{2} > h = 4\frac{1}{2}$, step 7(i) is completed, the candidate is undominated, and is to be "inserted between" columns 9 and 8 in order to honor (4.2).

During step 7(i) we keep track of column $J=j^*$ containing the greatest lower bound on $\underline{r}$ such that $B(m+1,j^*) < \underline{h}$. This is column $j^* = 7$. Thus, starting at column $B(m+3,j^*) = B(5,7) = 2$, we quickly trace through the payoffs via $B(m+3,J)$ and determine that column 9 holds the smallest payoff among all columns with $B(m+1,J) \geq \underline{h}$. Starting here we check if the candidate dominates column 9. It cannot, since $\beta_2 > B(2,9)$. Now, $B(m+3,9)$ leads us to column 8. Since $B(m+1,8) \geq \underline{h}$ and $B(0,8) = \underline{r}$, we must check again for dominance. We discover that column 8 is dominated by the candidate, and thus can be fathomed immediately. Furthermore, $B(m+3,8)$ leads to column 10 where $B(0,10) > \underline{r}$. *Therefore, we can stop the search here,* and step 7 (ii) is completed. The updated array appears below, listed in h order. The candidate has replaced the old fathomed column 8.

	1	3	6	5	4	2	7	9	8	10			
0	0	2	2.5	3	2.5	4	3.5	4.5	5	5.5			
1	0	2	2.5	2	1	3	1	4.5	3.5	3			
2	0	2	1.5	2.5	3	2	3.5	3.5	4.5	5.5			
3	0	2	2.5	2.5	3	3	3.5	4.5	4.5	5.5			
4	3	6	5	4	2	7	9	8	10	1			
5	3	6	4	7	5	9	2	8	10	1			
	β_1^0	β_1^1			β_1^2	β_1^3							

Row

The final B-list is given below, first as it appears in main memory, and then as it would appear in h order.

Row

	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	4	2	2.5	3	2.5	3.5	5	4.5	5.5	7	7.5	5.5
1	0	3	2	1	2	2.5	1	3.5	4.5	3	5	4	4.5
2	0	2	2	3	2.5	1.5	3.5	4.5	3.5	5.5	4.5	5.5	4.5
3	0	3	2	3	2.5	2.5	3.5	4.5	4.5	5.5	5	5.5	4.5
4	3	7	6	2	4	5	9	13	8	12	10	1	11
5	3	9	6	5	7	4	2	13	8	11	12	1	10

	1	3	6	5	4	2	7	9	8	13	11	10	12
0	0	2	2.5	3	2.5	4	3.5	4.5	5	5.5	7	5.5	7.5
1	0	2	2.5	2	1	3	1	4.5	3.5	4.5	5	3	4
2	0	2	1.5	2.5	3	2	3.5	3.5	4.5	4.5	4.5	5.5	5.5
3	0	2	2.5	2.5	3	3	3.5	4.5	4.5	4.5	5	5.5	5.5
4	3	6	5	4	2	7	9	8	13	11	10	12	1
5	3	6	4	7	5	9	2	8	13	10	12	11	1
	β_3^0	β_3^1											β_3^{12}

We note that of the (4)(4)(3) = 48 possible solutions, only 13 are feasible and undominated.

References

1. Bellman, R.E. and Dreyfus, S.E., "Dynamic Programming and the Reliability of Multicomponent Devices," Operations Research, 6 (1958), pp. 200-206

2. Bellman, R., "Some Applications of the Theory of Dynamic Programming--A Review," Operations Research, 2 (1954), pp. 275-288.

3. Bellman, R., "Notes on the Theory of Dynamic Programming IV--Maximization Over Discrete Sets," Nav. Res. Log. Quart., 3 (1956), pp. 67-70.

4. Bellman, R., "Comment on Dantzig's Paper on Discrete Variable Extremum Problems," Operations Research, 5 (1957), pp. 723-724.

5. Dantzig, G., "Discrete-Variable Extremum Problems," Operations Research, 5 (1957), pp. 266-277.

6. Denardo, E.V., and B.L. Fox, "Shortest Route Methods: 1. Reaching, Pruning, and Buckets," Operations Research, 27 (1979), pp. 161-186.

7. Dreyfus, S.E. and K.L. Prather, "Improved Algorithms for Knapsack Problems," Operations Research Center Report ORC 70-30, University of California-Berkeley (1970).

8. Faaland, B., "An Integer Programming Algorithm for Portfolio Selection," Management Science, 20 (1974), pp. 1376-1384.

9. Gilmore, P.C. and R.E. Gomory, "The Theory and Computation of Knapsack Functions," Operations Research, 14 (1966), pp 1045-1074.

10. Greenberg, H., "An Algorithm for the Computation of Knapsack Functions," Journal of Mathematical Analysis and Applications, 26 (1969), pp. 159-162.

11. Haymond R.E., "Discontinuities in the Optimal Return in Dynamic Programming," Journal of Mathematical Analysis and Applications, 30 (1970), pp. 639-644.

12. Karush, W., "A General Algorithm for the Optimal Distribution of Effort," Management Science, 9 (1962), pp. 50-72.

13. Ketelle, J.D., "Least-Cost Allocations of Reliability Investment," Operations Research, 10 (1962), pp. 249-265.

14. Lawler, E.L. and M.D. Bell, "A Method for Solving Discrete Optimization Problems," Operations Research, 14 (1966), pp. 1098-1112.

15. Loane, E.P., "An Algorithm to Solve Finite Separable Single-Constrained Optimization Problems," Operations Research, 16 (1968), pp. 1477-1493.

16. Marsten, R.E. and T.L. Morin, "A Hybrid Approach to Discrete Mathematical Programming," Mathematical Programming, 14 (1978), pp. 21-40.

17. Morin, T. L. and A.M.O. Esogbue, "The Imbedded State Space Approach to Reducing Dimensionality in Dynamic Programs of Higher Dimensions," Journal of Mathematical Analysis and Applications, 48 (1974), pp. 801-810.

18. Morin, T.L. and R.E. Marsten," An Algorithm for Nonlinear Knapsack Problems," Management Science, 22 (1976), pp. 1147-1158.

19. Nemhauser, G. L. and Z. Ullman, "Discrete Dynamic Programming and Capital Allocation," Management Science, 15 (1969), pp. 494-505.

20. Weingartner, H.M., "Capital Budgeting of Interrelated Projects: Survey and Synthesis," Management Science, 12 (1966), pp. 485-516.

21. Weingartner, H.M. and D.N. Ness, "Methods for the Solution of the Multi-Dimensional 0/1 Knapsack Problem," Operations Research, 15 (1967), pp. 83-103.

22. Yormark, J.S., "Accelerating Greenberg's Method for the Computation of Knapsack Functions," Journal of Mathematical Analysis and Applications, 49 (1975), pp. 629-637.

23. Yormark, J.S., "Resource Parametric Solutions for Nonlinear Multi-Dimensional Knapsack Problems," paper to be presented at XXIV International TIMS Meeting, Honolulu, Hawaii, June 1979.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (4) EFFICIENT FRONTIERS FOR NONLINEAR MULTIDIMENSIONAL KNAPSACK PROBLEMS,	5. TYPE OF REPORT & PERIOD COVERED (9) Technical / Rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Jonathan S. Yormark	8. CONTRACT OR GRANT NUMBER(s) (15) N00014-75-C-0733
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Finance & Business Economics University of Southern California Los Angeles, CA 90007	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR042-323
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Statistics and Probability Program, Code 436 Arlington, Virginia 22217	(11) 12. REPORT DATE January 1979 / 13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 26p.	15. SECURITY CLASS. (of this report) Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(14) USC-WP-2

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Nonlinear Knapsack Problem
Dynamic Programming
Integer Programming
Efficient Frontier

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An efficient recursive enumeration procedure is described for finding the complete family of efficient, or undominated, solutions for general separable multidimensional "knapsack" problems. A generalization of the Gilmore-Gomory procedure provides an alternative view of the Morin-Marsten method that admits an algorithm requiring significantly less computer storage and computational overhead. We illustrate the procedure with a highly nonlinear example.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102 LF 014 6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411 163